Truffle Interpreter Performance without the Holy Graal

April 2nd, 2022
Graal Workshop, CGO

Stefan Marr, Octave Larose, Sophie Kaleba, Chris Seaton

University of Kent

shopify

THE ROYAL SOCIETY

Job Ad

PostDoc Position

Project CaMELot: Catch and Mitigate
Event-Loop Concurrency Issues

Please get in touch!

University of Kent

https://stefan-marr.de/2021/02/open-postdoc-position-on-language-implementation-and-concurrency/

Truffle Interpreter Performance without the Holy Graal

shopify

University of Kent

April 2nd, 2022
Graal Workshop, CGO

Stefan Marr, Octave Larose, Sophie Kaleba, Chris Seaton

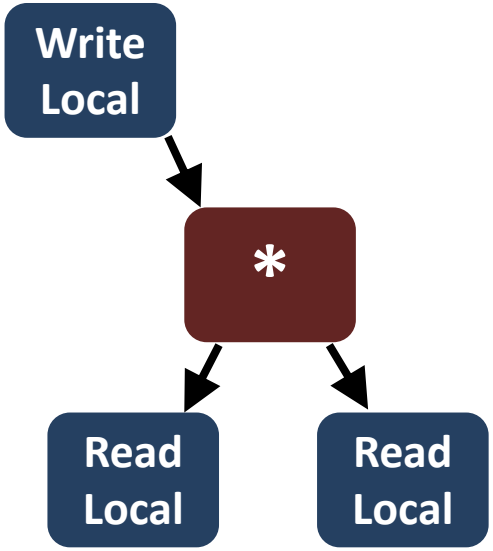
THE ROYAL SOCIETY

The Promise

Implement your language …

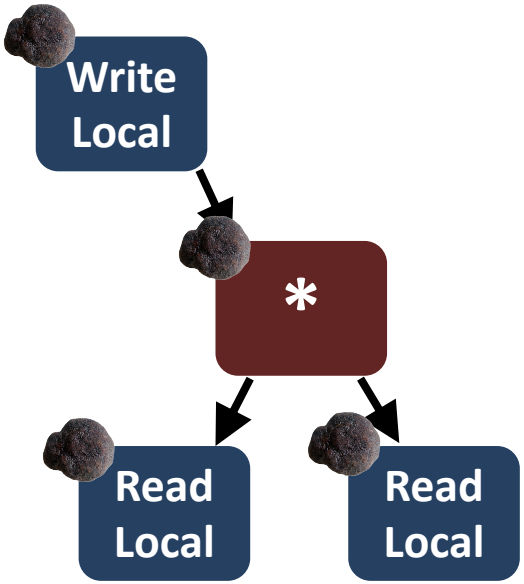
```
function mandelbrot() {
  let zr = 0.0;
  // while ... while ...
    zrzr = zr * zr;
  // ...
}
```

as an interpreter



The Promise

Sprinkle Truffle self-optimization on top



for instance, type profiling

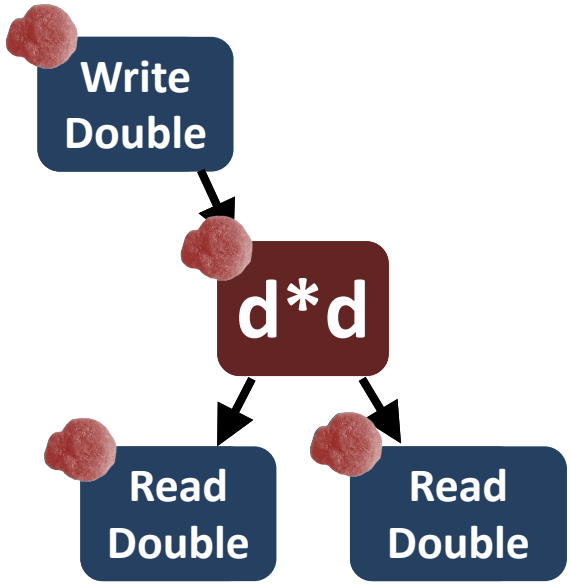
```
abstract class ReadLocal extends Node {
  private final slotIndex;
  @Specialization(
    guards = "frame.isDouble(slotIndex)")
  double doDouble(VirtualFrame frame) {
    return frame.getDouble(slotIndex);
  }
}
```

The Promise

Run your program

```
$ my-lang benchmark Mandelbrot 100
```

At run time, the interpreter specializes itself as needed

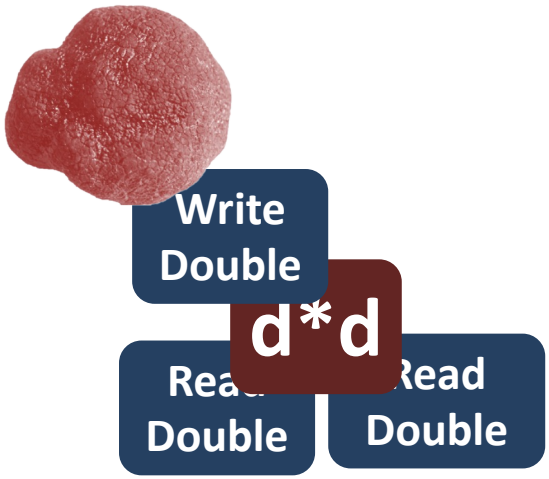


The Promise

Run your program

```
$ my-lang benchmark Mandelbrot 100
```

Apply Partial Evaluation during JIT-compilation



Execute the optimized code

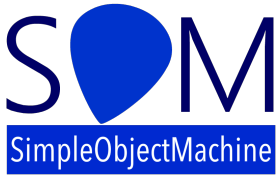
```
movapd …,…
mulsd  …,…
```

The Holy Graal

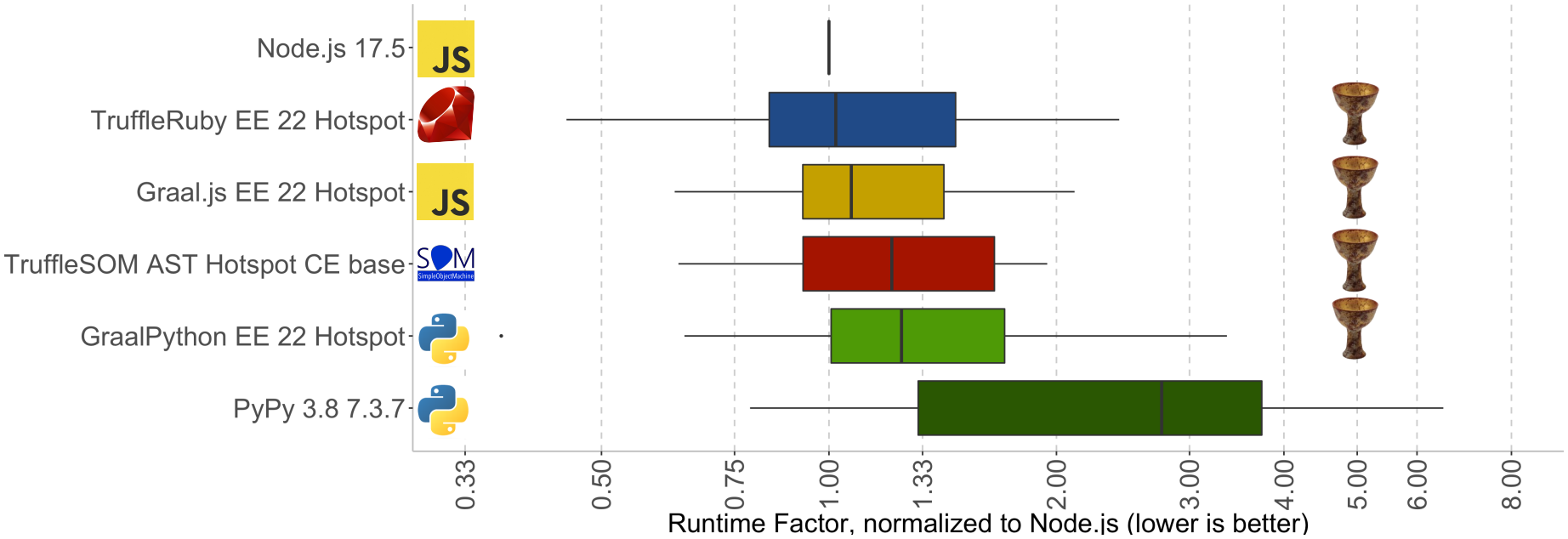
DOES THE PROMISE HOLD?

Are We Fast Yet

- Benchmarking a Common **Core Language**
- 9 micro benchmarks
- 5 not-quite so micro benchmarks
- 9 languages supported + some independent ports



The Promise Holds!



on the AreWeFastYet benchmarks,

Truffle-based languages are competitive with the state of the art

when relying on just-in-time compilation.

But!

Just-in-Time Compilers
need time to warm up...

Large Applications, Frequent Changes

> 100 million
lines of Hack/PHP

deployed every
75 minutes[1]

1. HHVM Jump-Start: Boosting Both Warmup and Steady-State Performance at Scale. Guilherme Ottoni, Bin Liu. CGO'21
2. https://shopify.engineering/automatic-deployment-at-shopify
3. https://instagram-engineering.com/continuous-deployment-at-instagram-1e18548f01d1
4. https://instagram-engineering.com/let-your-code-type-hint-itself-introducing-open-source-monkeytype-a855c7284881

Large Applications, Frequent Changes

> 100 million lines of Hack/PHP	2.8 million lines of Ruby
deployed every 75 minutes[1]	30-40 times a day (every 36-48min)[2]

1. HHVM Jump-Start: Boosting Both Warmup and Steady-State Performance at Scale. Guilherme Ottoni, Bin Liu. CGO'21
2. https://shopify.engineering/automatic-deployment-at-shopify
3. https://instagram-engineering.com/continuous-deployment-at-instagram-1e18548f01d1
4. https://instagram-engineering.com/let-your-code-type-hint-itself-introducing-open-source-monkeytype-a855c7284881

Large Applications, Frequent Changes

> 100 million lines of Hack/PHP	2.8 million lines of Ruby	> million lines of Python[4]
deployed every 75 minutes[1]	30-40 times a day (every 36-48min)[2]	30-50 times a day[3]

1. HHVM Jump-Start: Boosting Both Warmup and Steady-State Performance at Scale. Guilherme Ottoni, Bin Liu. CGO'21
2. https://shopify.engineering/automatic-deployment-at-shopify
3. https://instagram-engineering.com/continuous-deployment-at-instagram-1e18548f01d1
4. https://instagram-engineering.com/let-your-code-type-hint-itself-introducing-open-source-monkeytype-a855c7284881

The Cost of Change

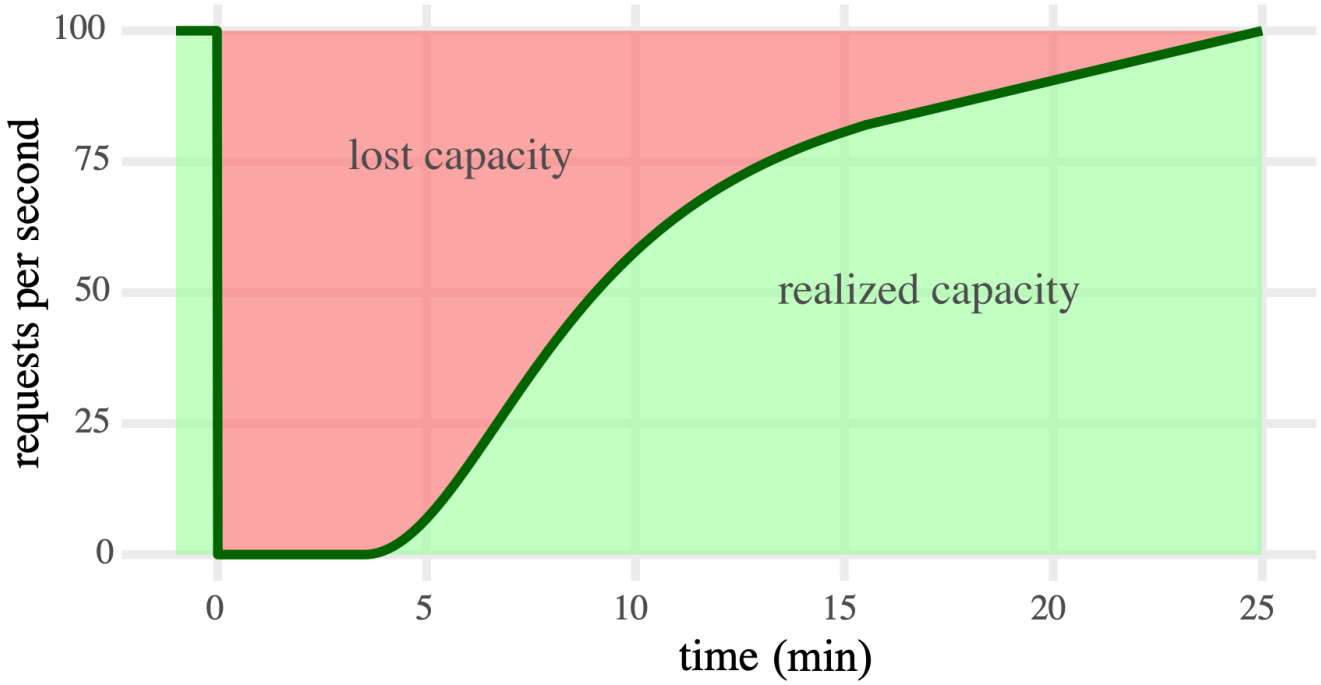


Fig. 2: Server capacity loss due to restart and warmup.

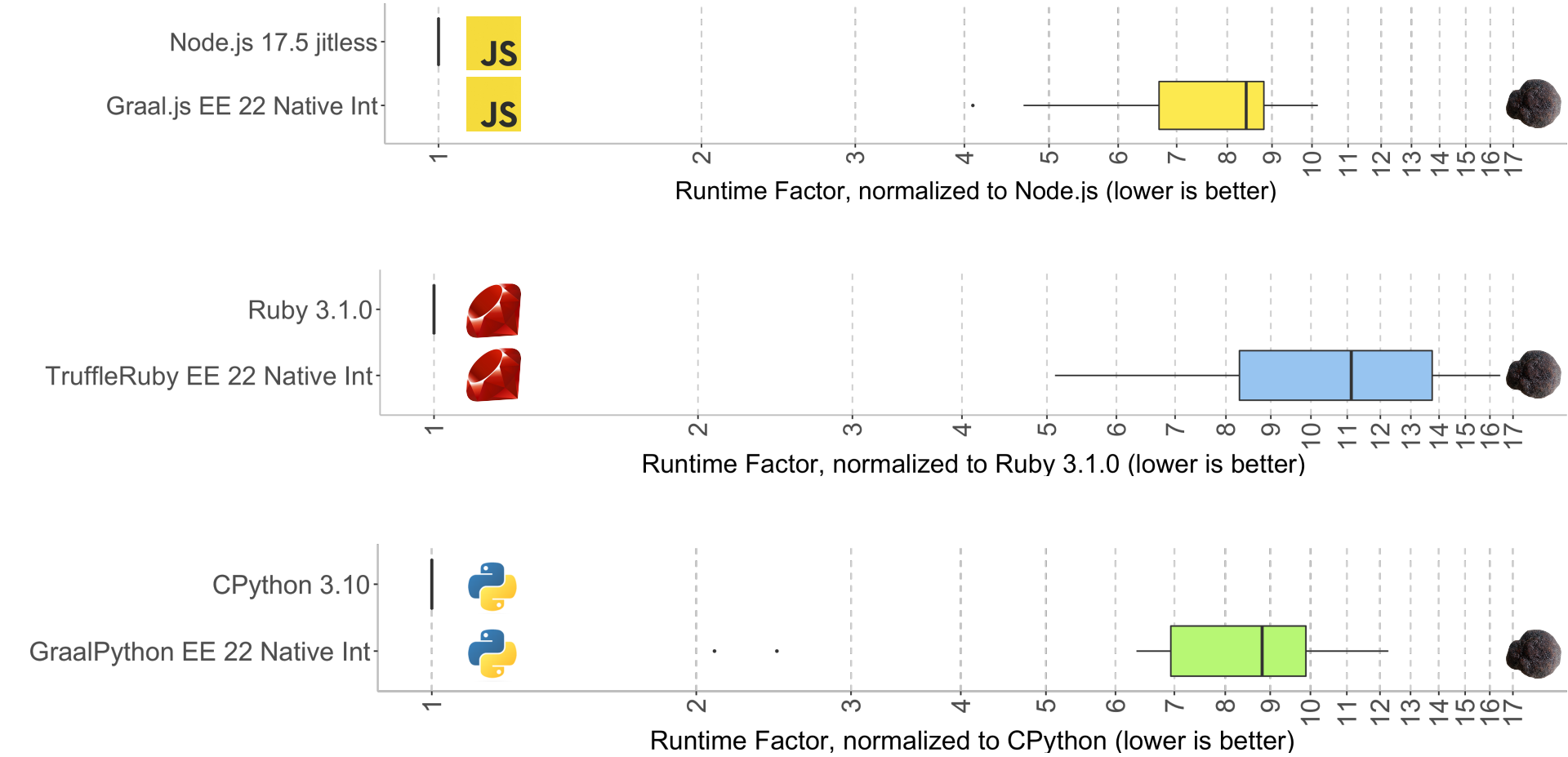
A Typical Web Server

HHVM Jump-Start: Boosting Both Warmup and Steady-State Performance at Scale.
Guilherme Ottoni, Bin Liu. CGO'21

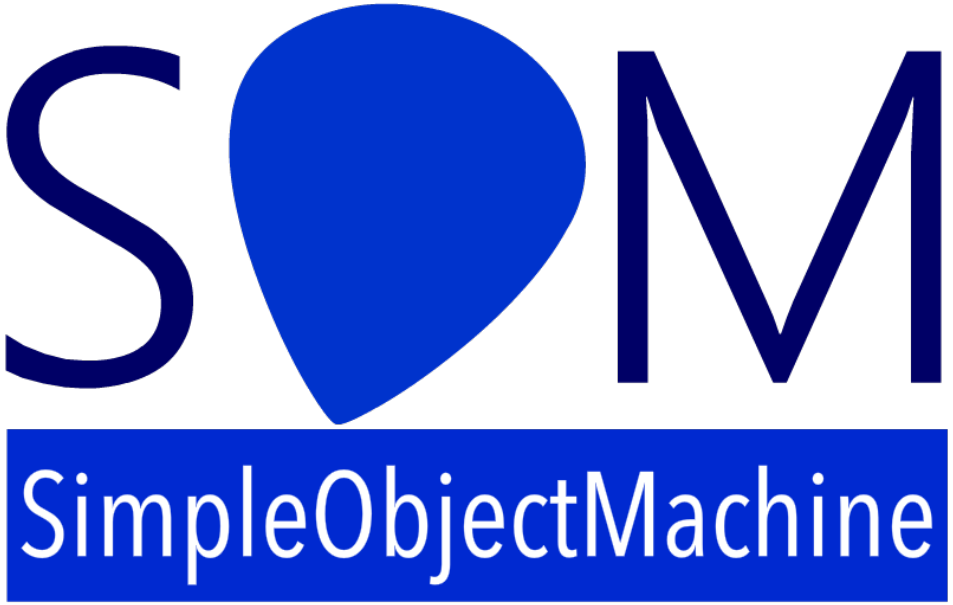
What if there’s no Graal?

for just-in-time compilation

How Fast are Truffle Interpreters?



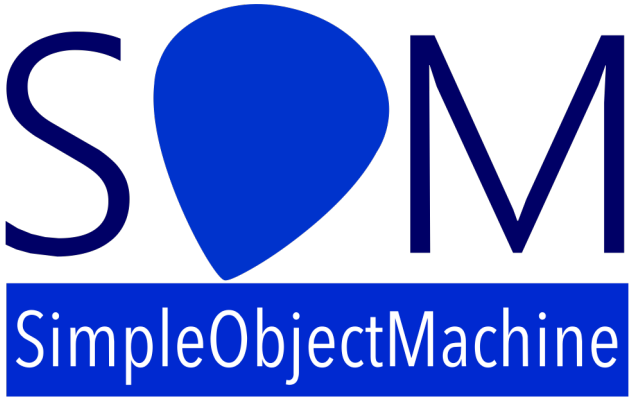
SOM: The Language use for our Experiments



som-st.github.io
@SOM_VMs

A Language For *Teaching* and *Research*

- A minimal object-oriented language
- Classes
  - Single inheritance
- Dynamic dispatch of methods (late binding)
- Non-local returns



Small enough to be understood

Big enough to be useful!

From 2001/2002

Kasper Verdich Lund
Lars Bak

SOM

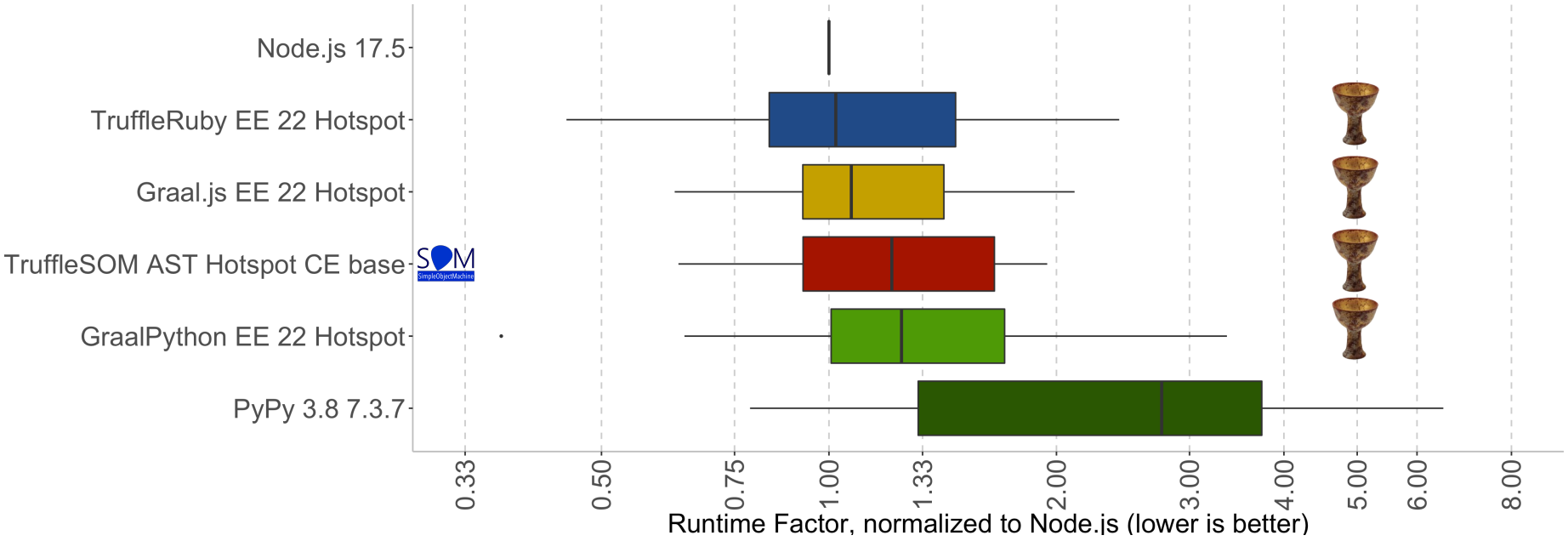
JsSOM

som-st.github.io
@SOM_VMs

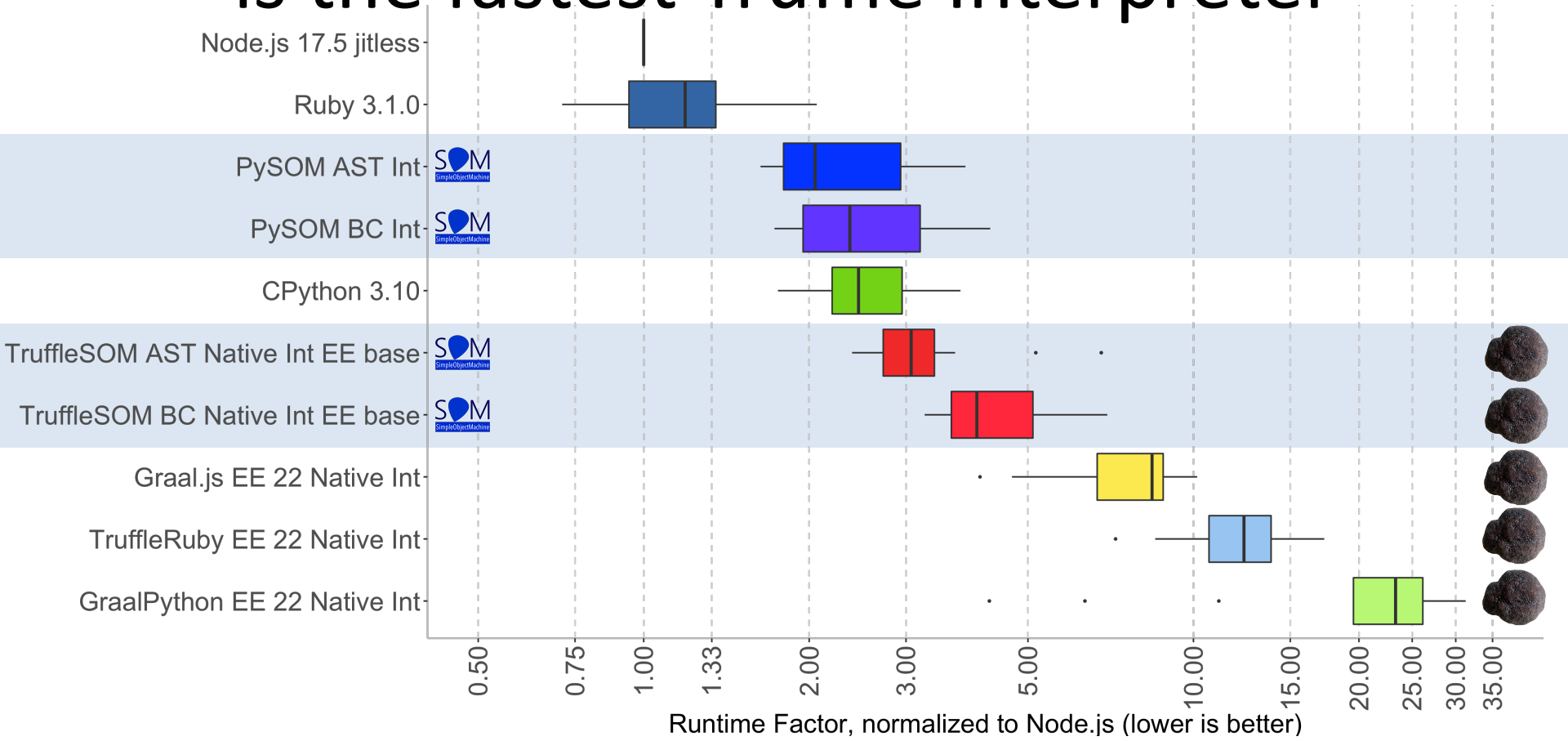
PySOM

GraalVM
TruffleSOM

But most of all, it's a suitable platform for experiments!



TruffleSOM
is the fastest Truffle Interpreter



Where do Truffle Interpreters spend their time?

Profiling the Interpreter

GreyObjectsWalker_walkGreyObjects	
FrameWithoutBoxing_constructor	
OptimizedCallTarget_profiledPERoot	
AbstractMessageSendNode_evaluateArguments	
SequenceNode_executeGeneric	
OptimizedCallTarget_profileArguments	
OptimizedCallTarget_executeRootNode	
OptimizedDirectCallNode_call	
GenericMessageSendNode_doPreEvaluated	90
CachedDispatchNode_executeDispatch	
CodeInfoDecoder_lookupCodeInfo	
ArgumentReadNode$LocalArgumentReadNode_executeGeneric	
FrameWithoutBoxing_verifyIndexedGet	
OptimizedCallTarget_callDirect	
UnmanagedMemoryUtil_copyLongsForward	
SubstrateArraycopySnippets_doArraycopy	
LocalVariableNodeFactory$LocalVariableWriteNodeGen_executeGeneric_long1	
OptimizedCallTarget_profileReturnValue	
LocalVariableNode$LocalVariableWriteNode_isLongKind	68
LocalVariableNodeFactory$LocalVariableReadNodeGen_executeLong	66
SubstrateOptimizedCallTargetInstalledCode_doInvoke	66
OptimizedAssumption_check	63
FrameWithoutBoxing_getObject	60
Method_execute	59
AbstractMessageSendNode_executeGeneric	58
OptimizedCallTarget_callBoundary	58
GraalRuntimeSupport_onLoopCount	55
OptimizedCallTarget_interpreterCall	54
FrameWithoutBoxing_getLong	54
AlignedChunkRememberedSet_walkDirtyObjects	54
FieldAccessorNode$ReadSpecializedFieldNode_hasExpectedLayout	52
FrameWithoutBoxing_setLong	50
FieldNode$FieldReadNode_executeGeneric	50
FrameWithoutBoxing_getTag	44

Profiling the Community Edition Native Images

Aggregate relative numbers over all benchmarks (no weighting)

Profiling the Interpreter

GreyObjectsWalker_walkGreyObjects	474
FrameWithoutBoxing_constructor	
OptimizedCallTarget_profiledPERoot	
AbstractMessageSendNode_evaluateArguments	
SequenceNode_executeGeneric	
OptimizedCallTarget_profileArguments	117
OptimizedCallTarget_executeRootNode	108
OptimizedDirectCallNode_call	98
GenericMessageSendNode_doPreEvaluated	90
CachedDispatchNode_executeDispatch	85
CodeInfoDecoder_lookupCodeInfo	79
ArgumentReadNode$LocalArgumentReadNode_executeGeneric	79
FrameWithoutBoxing_verifyIndexedGet	78
OptimizedCallTarget_callDirect	77
UnmanagedMemoryUtil_copyLongsForward	71
SubstrateArraycopySnippets_doArraycopy	71
LocalVariableNodeFactory$LocalVariableWriteNodeGen_executeGeneric_long1	70
OptimizedCallTarget_profileReturnValue	69
LocalVariableNode$LocalVariableWriteNode_isLongKind	68
LocalVariableNodeFactory$LocalVariableReadNodeGen_executeLong	66
SubstrateOptimizedCallTargetInstalledCode_doInvoke	66
OptimizedAssumption_check	63
FrameWithoutBoxing_getObject	60
Method_execute	59
AbstractMessageSendNode_executeGeneric	58
OptimizedCallTarget_callBoundary	58
GraalRuntimeSupport_onLoopCount	55
OptimizedCallTarget_interpreterCall	54
FrameWithoutBoxing_getLong	54
AlignedChunkRememberedSet_walkDirtyObjects	54
FieldAccessorNode$ReadSpecializedFieldNode_hasExpectedLayout	52
FrameWithoutBoxing_setLong	50
FieldNode$FieldReadNode_executeGeneric	50
FrameWithoutBoxing_getTag	44

CE GC implementation

Profiling the Interpreter

GreyObjectsWalker_walkGreyObjects	474
FrameWithoutBoxing_constructor	
OptimizedCallTarget_profiledPERoot	
AbstractMessageSendNode_evaluateArguments	
SequenceNode_executeGeneric	132
OptimizedCallTarget_profileArguments	117
OptimizedCallTarget_executeRootNode	108
OptimizedDirectCallNode_call	98
GenericMessageSendNode_doPreEvaluated	90
CachedDispatchNode_executeDispatch	
CodeInfoDecoder_lookupCodeInfo	
ArgumentReadNode$LocalArgumentReadNode_executeGeneric	
FrameWithoutBoxing_verifyIndexedGet	
OptimizedCallTarget_callDirect	
UnmanagedMemoryUtil_copyLongsForward	
SubstrateArraycopySnippets_doArraycopy	
LocalVariableNodeFactory$LocalVariableWriteNodeGen_executeGeneric_long1	
OptimizedCallTarget_profileReturnValue	
LocalVariableNode$LocalVariableWriteNode_isLongKind	
LocalVariableNodeFactory$LocalVariableReadNodeGen_executeLong	
SubstrateOptimizedCallTargetInstalledCode_doInvoke	
OptimizedAssumption_check	63
FrameWithoutBoxing_getObject	60
Method_execute	59
AbstractMessageSendNode_executeGeneric	58
OptimizedCallTarget_callBoundary	58
GraalRuntimeSupport_onLoopCount	55
OptimizedCallTarget_interpreterCall	54
FrameWithoutBoxing_getLong	54
AlignedChunkRememberedSet_walkDirtyObjects	54
FieldAccessorNode$ReadSpecializedFieldNode_hasExpectedLayout	52
FrameWithoutBoxing_setLong	50
FieldNode$FieldReadNode_executeGeneric	50
FrameWithoutBoxing_getTag	44

Truffle frames and call overhead

Tried optimizing it, benchmarks are not impressed.
And important for JITed Performance.

Profiling the Interpreter

GreyObjectsWalker_walkGreyObjects	474
FrameWithoutBoxing_constructor	226
OptimizedCallTarget_profiledPERoot	175
AbstractMessageSendNode_evaluateArguments	
SequenceNode_executeGeneric	
OptimizedCallTarget_profileArguments	
OptimizedCallTarget_executeRootNode	
OptimizedDirectCallNode_call	98
GenericMessageSendNode_doPreEvaluated	90
CachedDispatchNode_executeDispatch	85
CodeInfoDecoder_lookupCodeInfo	79
ArgumentReadNode$LocalArgumentReadNode_executeGeneric	79
FrameWithoutBoxing_verifyIndexedGet	78
OptimizedCallTarget_callDirect	77
UnmanagedMemoryUtil_copyLongsForward	71
SubstrateArraycopySnippets_doArraycopy	71
LocalVariableNodeFactory$LocalVariableWriteNodeGen_executeGeneric_long1	70
OptimizedCallTarget_profileReturnValue	69
LocalVariableNode$LocalVariableWriteNode_isLongKind	68
LocalVariableNodeFactory$LocalVariableReadNodeGen_executeLong	66
SubstrateOptimizedCallTargetInstalledCode_doInvoke	66
OptimizedAssumption_check	63
FrameWithoutBoxing_getObject	60
Method_execute	59
AbstractMessageSendNode_executeGeneric	58
OptimizedCallTarget_callBoundary	58
GraalRuntimeSupport_onLoopCount	55
OptimizedCallTarget_interpreterCall	54
FrameWithoutBoxing_getLong	54
AlignedChunkRememberedSet_walkDirtyObjects	54
FieldAccessorNode$ReadSpecializedFieldNode_hasExpectedLayout	52
FrameWithoutBoxing_setLong	50
FieldNode$FieldReadNode_executeGeneric	50
FrameWithoutBoxing_getTag	44

Evaluating Arguments for Method Calls

Profiling the Interpreter

GreyObjectsWalker_walkGreyObjects	474
FrameWithoutBoxing_constructor	226
OptimizedCallTarget_profiledPERoot	175
AbstractMessageSendNode_evaluateArguments	
SequenceNode_executeGeneric	
OptimizedCallTarget_profileArguments	
OptimizedCallTarget_executeRootNode	
OptimizedDirectCallNode_call	98
GenericMessageSendNode_doPreEvaluated	90
CachedDispatchNode_executeDispatch	85
CodeInfoDecoder_lookupCodeInfo	79
ArgumentReadNode$LocalArgumentReadNode_executeGeneric	79
FrameWithoutBoxing_verifyIndexedGet	78
OptimizedCallTarget_callDirect	77
UnmanagedMemoryUtil_copyLongsForward	71
SubstrateArraycopySnippets_doArraycopy	71
LocalVariableNodeFactory$LocalVariableWriteNodeGen_executeGeneric_long1	70
OptimizedCallTarget_profileReturnValue	69
LocalVariableNode$LocalVariableWriteNode_isLongKind	68
LocalVariableNodeFactory$LocalVariableReadNodeGen_executeLong	66
SubstrateOptimizedCallTargetInstalledCode_doInvoke	66
OptimizedAssumption_check	63
FrameWithoutBoxing_getObject	60
Method_execute	59
AbstractMessageSendNode_executeGeneric	58
OptimizedCallTarget_callBoundary	58
GraalRuntimeSupport_onLoopCount	55
OptimizedCallTarget_interpreterCall	54
FrameWithoutBoxing_getLong	54
AlignedChunkRememberedSet_walkDirtyObjects	54
FieldAccessorNode$ReadSpecializedFieldNode_hasExpectedLayout	52
FrameWithoutBoxing_setLong	50
FieldNode$FieldReadNode_executeGeneric	50
FrameWithoutBoxing_getTag	44

Very Generic Things™

Profiling the Interpreter

GreyObjectsWalker_walkGreyObjects	474
FrameWithoutBoxing_constructor	226
OptimizedCallTarget_profiledPERoot	175
AbstractMessageSendNode_evaluateArguments	148
SequenceNode_executeGeneric	132
OptimizedCallTarget_profileArguments	117
OptimizedCallTarget_executeRootNode	108
OptimizedDirectCallNode_call	98
GenericMessageSendNode_doPreEvaluated	
CachedDispatchNode_executeDispatch	
CodeInfoDecoder_lookupCodeInfo	
ArgumentReadNode$LocalArgumentReadNode_executeGeneric	
FrameWithoutBoxing_verifyIndexedGet	78
OptimizedCallTarget_callDirect	77
UnmanagedMemoryUtil_copyLongsForward	71
SubstrateArraycopySnippets_doArraycopy	71
LocalVariableNodeFactory$LocalVariableWriteNodeGen_executeGeneric_long1	70
OptimizedCallTarget_profileReturnValue	69
LocalVariableNode$LocalVariableWriteNode_isLongKind	68
LocalVariableNodeFactory$LocalVariableReadNodeGen_executeLong	66
SubstrateOptimizedCallTargetInstalledCode_doInvoke	66
OptimizedAssumption_check	63
FrameWithoutBoxing_getObject	60
Method_execute	59
AbstractMessageSendNode_executeGeneric	58
OptimizedCallTarget_callBoundary	58
GraalRuntimeSupport_onLoopCount	55
OptimizedCallTarget_interpreterCall	54
FrameWithoutBoxing_getLong	54
AlignedChunkRememberedSet_walkDirtyObjects	54
FieldAccessorNode$ReadSpecializedFieldNode_hasExpectedLayout	52
FrameWithoutBoxing_setLong	50
FieldNode$FieldReadNode_executeGeneric	50
FrameWithoutBoxing_getTag	44

Method Lookup Caches

Profiling the Interpreter

GreyObjectsWalker_walkGreyObjects	474
FrameWithoutBoxing_constructor	226
OptimizedCallTarget_profiledPERoot	175
AbstractMessageSendNode_evaluateArguments	148
SequenceNode_executeGeneric	132
OptimizedCallTarget_profileArguments	117
OptimizedCallTarget_executeRootNode	108
OptimizedDirectCallNode_call	98
GenericMessageSendNode_doPreEvaluated	90
CachedDispatchNode_executeDispatch	85
CodeInfoDecoder_lookupCodeInfo	
ArgumentReadNode$LocalArgumentReadNode_executeGeneric	
FrameWithoutBoxing_verifyIndexedGet	
OptimizedCallTarget_callDirect	
UnmanagedMemoryUtil_copyLongsForward	71
SubstrateArraycopySnippets_doArraycopy	71
LocalVariableNodeFactory$LocalVariableWriteNodeGen_executeGeneric_long1	70
OptimizedCallTarget_profileReturnValue	69
LocalVariableNode$LocalVariableWriteNode_isLongKind	68
LocalVariableNodeFactory$LocalVariableReadNodeGen_executeLong	66
SubstrateOptimizedCallTargetInstalledCode_doInvoke	66
OptimizedAssumption_check	63
FrameWithoutBoxing_getObject	60
Method_execute	59
AbstractMessageSendNode_executeGeneric	58
OptimizedCallTarget_callBoundary	58
GraalRuntimeSupport_onLoopCount	55
OptimizedCallTarget_interpreterCall	54
FrameWithoutBoxing_getLong	54
AlignedChunkRememberedSet_walkDirtyObjects	54
FieldAccessorNode$ReadSpecializedFieldNode_hasExpectedLayout	52
FrameWithoutBoxing_setLong	50
FieldNode$FieldReadNode_executeGeneric	50
FrameWithoutBoxing_getTag	44

Reading method arguments

Profiling the Interpreter

GreyObjectsWalker_walkGreyObjects	474
FrameWithoutBoxing_constructor	226
OptimizedCallTarget_profiledPERoot	175
AbstractMessageSendNode_evaluateArguments	148
SequenceNode_executeGeneric	132
OptimizedCallTarget_profileArguments	117
OptimizedCallTarget_executeRootNode	108
OptimizedDirectCallNode_call	98
GenericMessageSendNode_doPreEvaluated	90
CachedDispatchNode_executeDispatch	85
CodeInfoDecoder_lookupCodeInfo	79
ArgumentReadNode$LocalArgumentReadNode_executeGeneric	
FrameWithoutBoxing_verifyIndexedGet	
OptimizedCallTarget_callDirect	
UnmanagedMemoryUtil_copyLongsForward	
SubstrateArraycopySnippets_doArraycopy	
LocalVariableNodeFactory$LocalVariableWriteNodeGen_executeGeneric_long1	70
OptimizedCallTarget_profileReturnValue	69
LocalVariableNode$LocalVariableWriteNode_isLongKind	68
LocalVariableNodeFactory$LocalVariableReadNodeGen_executeLong	66
SubstrateOptimizedCallTargetInstalledCode_doInvoke	66
OptimizedAssumption_check	63
FrameWithoutBoxing_getObject	60
Method_execute	59
AbstractMessageSendNode_executeGeneric	58
OptimizedCallTarget_callBoundary	58
GraalRuntimeSupport_onLoopCount	55
OptimizedCallTarget_interpreterCall	54
FrameWithoutBoxing_getLong	54
AlignedChunkRememberedSet_walkDirtyObjects	54
FieldAccessorNode$ReadSpecializedFieldNode_hasExpectedLayout	52
FrameWithoutBoxing_setLong	50
FieldNode$FieldReadNode_executeGeneric	50
FrameWithoutBoxing_getTag	44

Accessing local variables in Truffle frames

Profiling the Interpreter

GreyObjectsWalker_walkGreyObjects	474
FrameWithoutBoxing_constructor	226
OptimizedCallTarget_profiledPERoot	175
AbstractMessageSendNode_evaluateArguments	148
SequenceNode_executeGeneric	132
OptimizedCallTarget_profileArguments	117
OptimizedCallTarget_executeRootNode	108
OptimizedDirectCallNode_call	98
GenericMessageSendNode_doPreEvaluated	90
CachedDispatchNode_executeDispatch	85
CodeInfoDecoder_lookupCodeInfo	**79**
ArgumentReadNode$LocalArgumentReadNode_executeGeneric	**79**
FrameWithoutBoxing_verifyIndexedGet	**78**
OptimizedCallTarget_callDirect	**77**
UnmanagedMemoryUtil_copyLongsForward	**71**
SubstrateArraycopySnippets_doArraycopy	**71**
LocalVariableNodeFactory$LocalVariableWriteNodeGen_executeGeneric_long1	**70**
OptimizedCallTarget_profileReturnValue	**69**
LocalVariableNode$LocalVariableWriteNode_isLongKind	**68**
LocalVariableNodeFactory$LocalVariableReadNodeGen_executeLong	**66**
SubstrateOptimizedCallTargetInstalledCode_doInvoke	**66**
OptimizedAssumption_check	**63**
FrameWithoutBoxing_getObject	**60**
Method_execute	**59**
AbstractMessageSendNode_executeGeneric	**58**
OptimizedCallTarget_callBoundary	**58**
GraalRuntimeSupport_onLoopCount	**55**
OptimizedCallTarget_interpreterCall	**54**
FrameWithoutBoxing_getLong	**54**
AlignedChunkRememberedSet_walkDirtyObjects	**54**
FieldAccessorNode$ReadSpecializedFieldNode_hasExpectedLayou	**52**
FrameWithoutBoxing_setLong	**50**
FieldNode$FieldReadNode_executeGeneric	**50**
FrameWithoutBoxing_getTag	**44**

A very long tail…

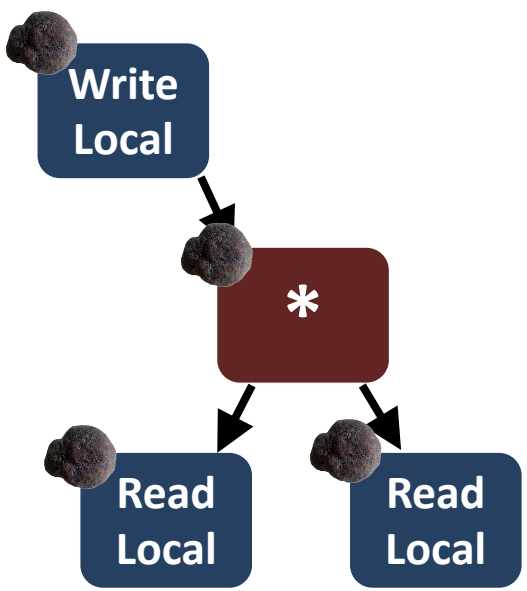
Conclusion:

It goes everywhere…

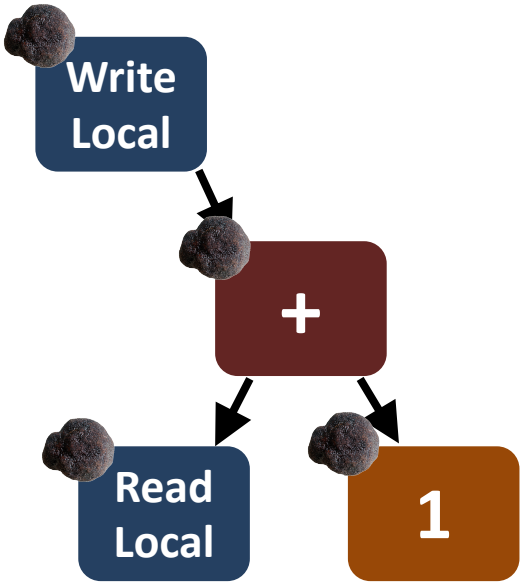
Death by a thousand paper cuts

Let’s do Some Experiments!

Common Node Patterns



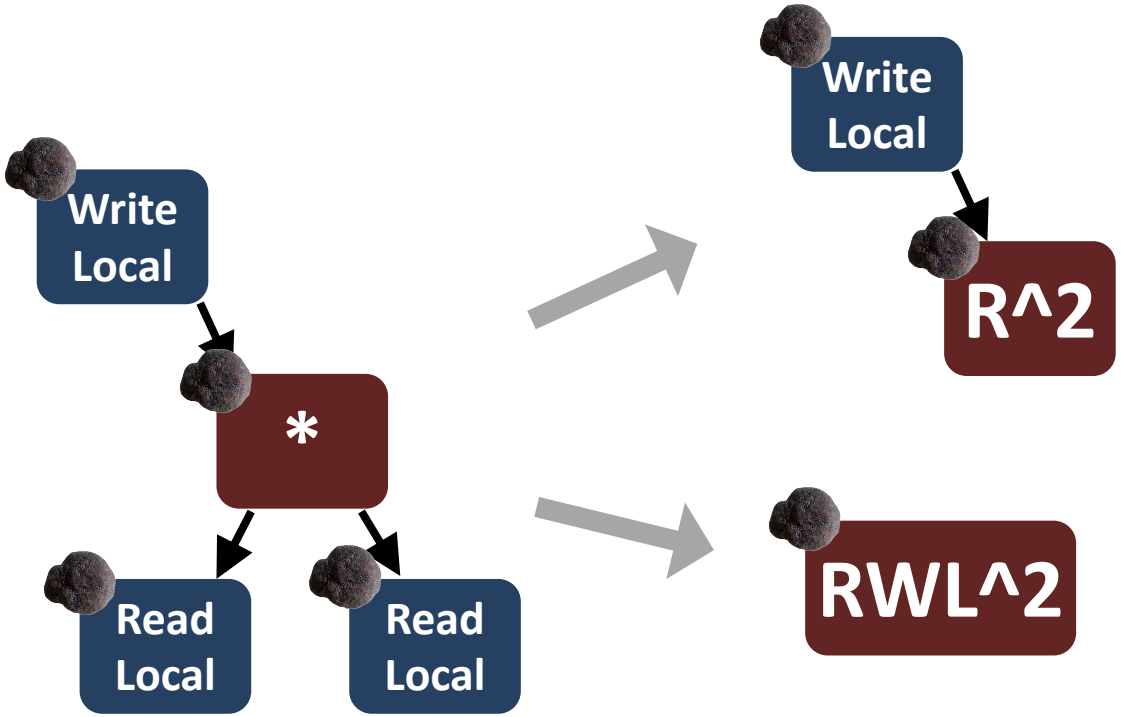

```
zrzr = zr * zr
```




```
i = i + 1
```

- We built a tool to identify those
- Based on static and dynamic usage statistics

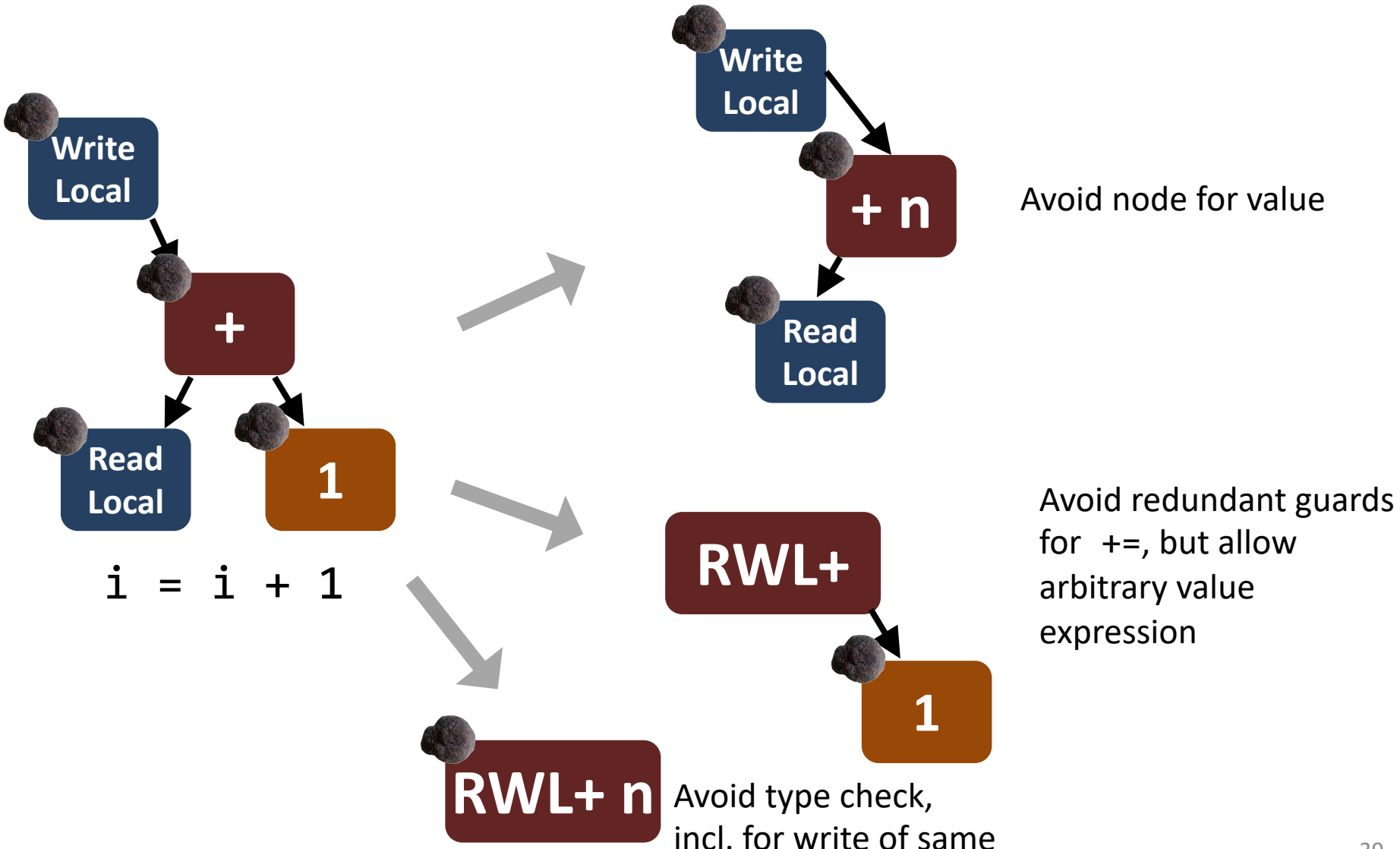
Create Square Super Nodes



Avoid redundant read and type check

Avoid redundant read and type check, incl. for write of same location

Create Increment Super Nodes



18 Super Nodes in Total

Variations of

- Method activation with `this` as argument
  - `this.method(arg1, arg2)`
- Increment/decrement operations
  - `var += n`
- Compute square
  - `var = d * d`
- String equality with constant
  - `var == 'constant'`

Very “focused” experiment

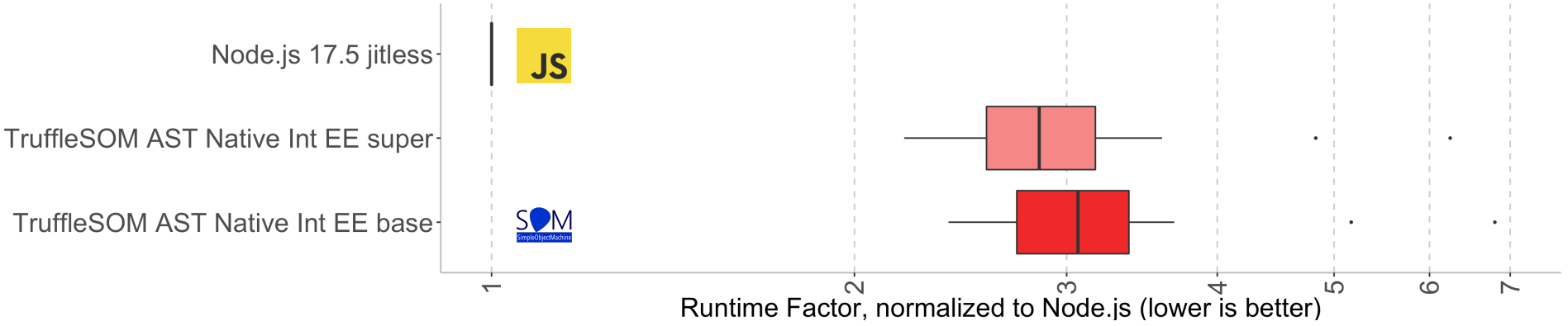
Results for Super Nodes

	median time in ms	time diff %
DeltaBlue	39.53	-2
GraphSearch	27.48	-3
Json	58.84	-14
NBody	33.62	-5
PageRank	24.94	-11
Richards	250.72	-11

- Optimizing small patterns can give a nice gain
- But not generalizable…

	median time in ms	time diff %
Bounce	47.07	-11
Dispatch	36.34	-19
Fibonacci	49.37	-11
FieldLoop	19.78	-56
Loop	43.69	-56
Mandelbrot	30.52	-26
Permute	77.10	-10
Queens	45.96	-6
QuickSort	41.27	-9
Recurse	43.65	-10
Sieve	34.31	-15
Sum	32.81	-34
Test	184.28	-6
Towers	28.55	-10
WhileLoop	29.98	-29

Big Picture “Super Nodes” on AWFY



- One word: disappointing
- Very “targeted”
- Limited overall impact

Four Patterns Required 18 Different Nodes!

- Method activation with `this` as argument
  - `this.method(arg1, arg2)`
- Increment/decrement operations
  - `var += n`
- Compute square
  - `var = d * d`
- String equality with constant
  - `var == 'constant'`

Node Explosion!
Code Explosion!
Not Maintainable!

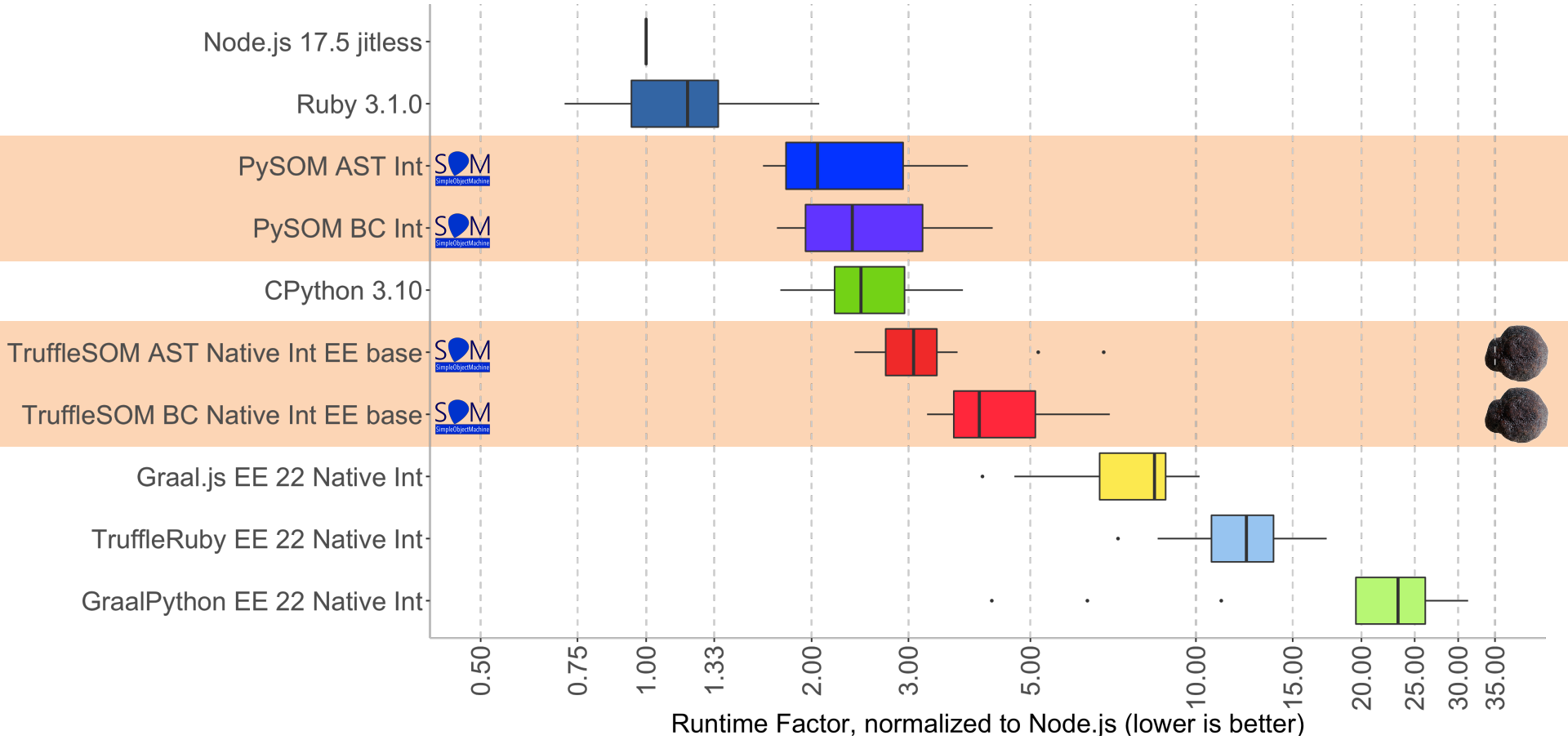
What to do about it?

How About Bytecodes?

- More compact than objects
- No GC overhead
- Perhaps smaller dispatch overhead?

```
HALT
DUP
PUSH_LOCAL
PUSH_ARGUMENT
PUSH_FIELD
PUSH_BLOCK
PUSH_CONSTANT
PUSH_GLOBAL
POP
POP_LOCAL
POP_ARGUMENT
POP_FIELD
SEND
SUPER_SEND
RETURN_LOCAL
RETURN_NON_LOCAL
```

My AST Interpreters Beat My Bytecode Interpreters



Except for One Benchmark

```
loadAllCodeFiles();

function benchmark() {
  System.gc();
}
```

15KLOC

	CE	EE G1
AST	150ms	140ms
BC	61ms	110ms
	-60%	-21%

Mini Bytecodes for Specific Use Cases

- List benchmark
  - Small method with a trivial loop
- CD benchmark
  - Small compare method, comparing numbers
- Random.next()
  - `seed = ((seed * 1309) + 13849) & 65535;`
    `return seed;`
- **Initializer/constructor methods**

Initializer/Constructor Methods

- They are fancy setters


```
store-arg1          into field
store-arg2          into field
store-arg3          into field
store-arg4          into field
store-arg5          into field
store-0             into field
store-1             into field

store-true          into field
store-false         into field
store-nil           into field
store-int           into field
store-object-const into field
```

Benefit of Mini Bytecode Interpreters

		time diff in %
List	7.60	-29
CD	394.82	-12
Storage	42.51	-8
Initializers	62.24	-11
Initializers	42.32	-32

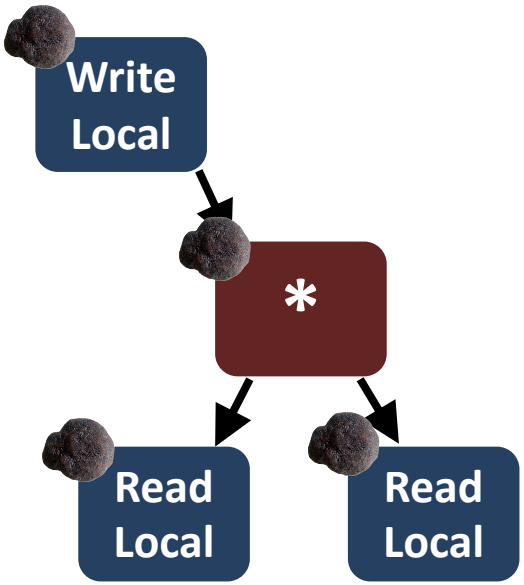
Recognize methods as trivial, avoid call overhead

https://rebench.stefan-marr.de/compare/TruffleSOM/31c5de79a71c628b0325c1fcb47b15bbe810245d/b70ceb0b4c6b10313e9784dc8231cb055c81e3d3

Avoiding overhead of virtual dispatch in AST

“BEST CASE” EXPERIMENT

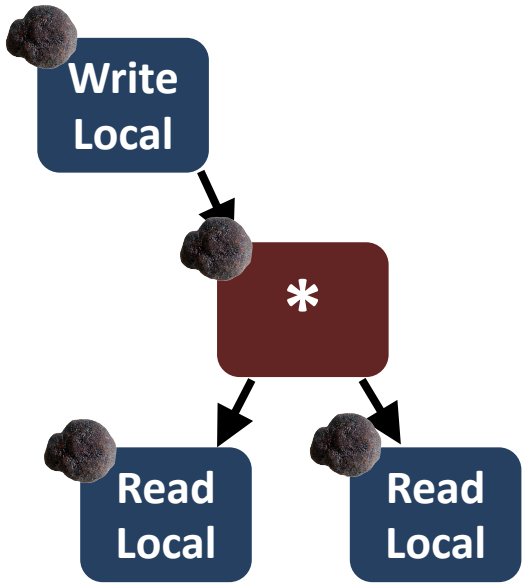
“Best Case”




```
abstract class ReadLocal extends Node {
  private final slotIndex;
  @Specialization(
    guards = "frame.isDouble(slotIndex)")
  double doDouble(VirtualFrame frame) {
    return frame.getDouble(slotIndex);
  }
}
```

```
class WriteLocal extends Expr {
  Object execute(VFrame f) { /*…*/ }
}
class Multiply extends Expr {
  Object execute(VFrame f) { /*…*/ }
}
class ReadLocal extends Expr {
  Object execute(VFrame f) { /*…*/ }
}
```

“Best Case”



- Method-level optimizations
- Assume
  - types are known
  - primitive operations can be inlined
  - Frame access only needed around loop boundaries
- But using existing nodes for dispatch, field+array access

```
frame.setDouble(zrzrSlot, zr * zr); // zrzr = zr * zr

try {
  frame.setLong(zSlot, Math.addExact
} catch (ArithmeticException e) {
  CompilerDirectives.transferToInter
  // ...
}
```

Caveat: Not Best Best Case
too optimistic and
too pessimistic!

Results for “Best Case” Method-level Optimizations

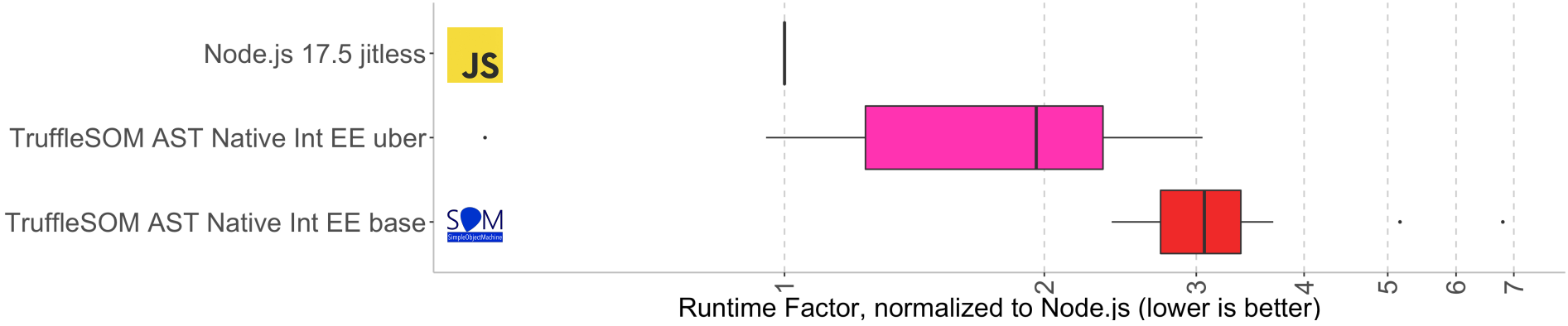
	median time in ms	time diff %
DeltaBlue	33.61	-16
Json	40.06	-41
NBody	33.43	-6
Richards	206.41	-27

	median time in ms	time diff %
Bounce	24.39	-54
List	3.85	-73
Mandelbrot	4.72	-89
Permute	32.37	-62
Queens	36.16	-26
Sieve	17.05	-58
Towers	21.80	-31

- Applied to SOM’s benchmarks
- Take 1-4 top methods from profile
- Convert to an “Über Node”
- Results: mixed

Limitation: Still reusing too many field/array/… access nodes to ensure semantics

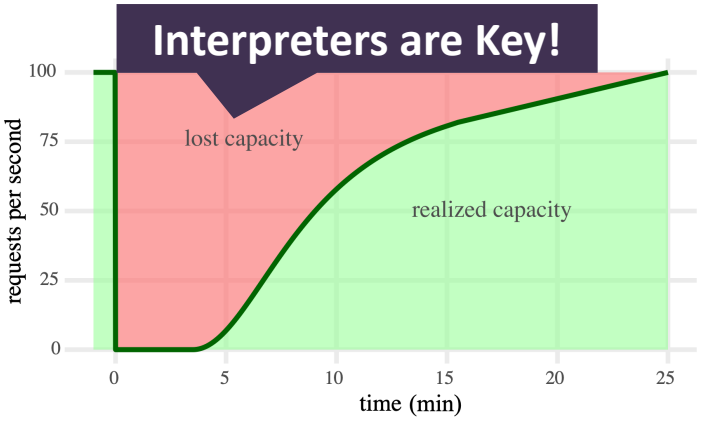
Big Picture “Best Case” on AWFY

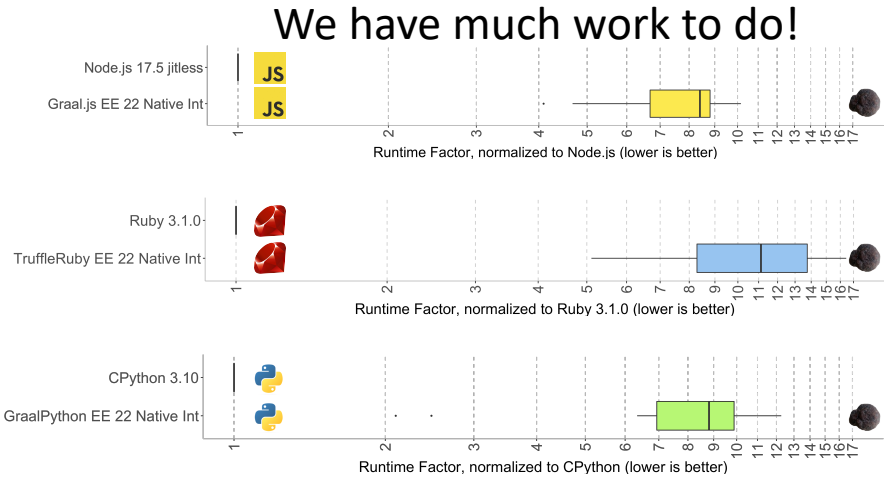


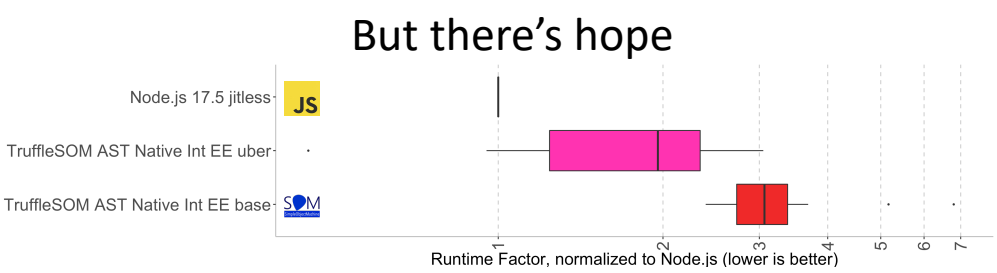
- Makes a difference, on some benchmarks
- Not sufficient to
  - avoid “AST”-node dispatch
  - avoid primitive type guards

WRAP UP

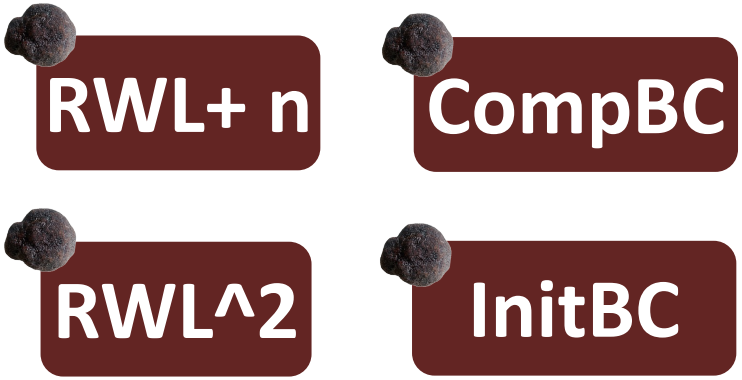
Truffle Interpreters can be faster than they currently are!







"Best Case" Experiment
shows there's room for improvements



SuperNodes and
Mini Bytecode Interpreters
may get us some of the way